Assistive Camera Control

ECE-405

SD1214 Members:

Steven Goldade
Lance Larsen
David Pinewski

Client:

Mark Coppin

ECE Advisor:

Dr. Green

TABLE OF CONTENTS

BACKGROUND

Our design project is an assistive camera device. This device is designed to help a person to easily control the zoom, pan and tilt of a DSLR camera. The camera will be mounted on the device, and will be controlled by a Java application on a computer through a switch interface/scanning system. With our device and software, the client will be able to control all aspects of the camera through her computer, including pan, tilt, zoom, and various settings.

REQUIREMENTS

ZOOM DRIVE SYSTEM

- Able to turn lens clockwise and counterclockwise
- Able to fully zoom in and out with a quarter turn of the lens
- Variable speed
- Controlled through the client's MacBook Pro
- Must fit 70.5 x 74mm lens

PAN/TILT DRIVE SYSTEM

- Pan at least +/- 45 degrees from center position
- Variable speed
- Controlled through the client's MacBook Pro

SOFTWARE

- Able to control zoom
- Able to support pan and tilt
- Able to support variable speed capabilities
- Able to create programed sequences for camera to follow

POWER

- Devices will be powered from the client's on board power supply
- Devices will have fault and overload protection

Device Overview

The Mechanicals of the device are broken down into three subsystems: pan, tilt, and zoom. These sub-systems are fastened together with the camera mounted on top of the zoom system. Each sub-system has a digital servomotor associated with it, and our specifically designed microcontroller board controls each servo.

Pan System

For the pan system, we are using an SPG-400BM pan system made by Servo City. We modified the SPG-400BM to have a 2:1 gear ratio and removed the external potentiometer in order to be more reliable. We are driving the pan system with a HS-5685MH, high voltage servo. The pan system is shown below in figure 1.1

Figure 1.1

Tilt System

For the tilt system, we are using an SPT-400 tilt system made by Servo City. We modified the SPT-400 to have a 7:1 gear ratio. We are driving the tilt system with a HS-7950TH, high voltage and torque servo. The tilt system, which mounts to the large gear of the pan system, is shown below in figure 2.1

Figure 2.1

ZOOM SYSTEM

The zoom system was designed by us, and machined out of aluminum by the IME department. The system is made up of two pieces of aluminum. The larger piece, which the camera mounts to, attaches to the top plate of the tilt system, and the smaller piece, which the servo mounts to, attaches to the larger piece with quarter inch bolts to dual slots on the plate. The slots allow the servo gear to be adjusted to mesh with the follow focus gear ring on the camera lens. Figure 3.1 below is a CAD rendering of the zoom system, showing how the camera and servo mount on the zoom system plates.

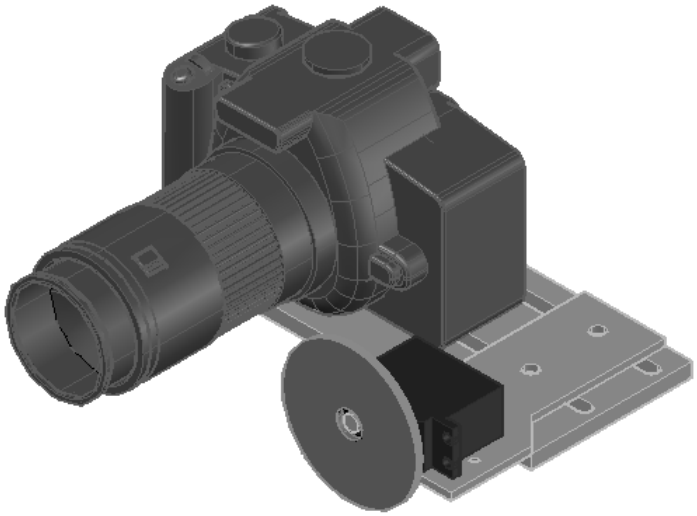


Figure 3.1

CONTROLLER

We designed a control board with all surface mount components to allow us to shrink it down in order to fit under the pan system. The control board features a PIC 18F series microcontroller and a FTDI FT232RL USB-UART converter, which allows us to send commands over USB to the microcontroller. The microcontroller outputs an independent, variable duty cycle, pulse train to each servo. The width of the pulse is what determines the angle of the servo output shaft. The control board components are powered off of USB, and the servos are powered from an external power supply, which connects to pins on the control board. The block diagram of the controller system is shown below in figure 4.1.

HARDWARE BLOCK DIAGRAM

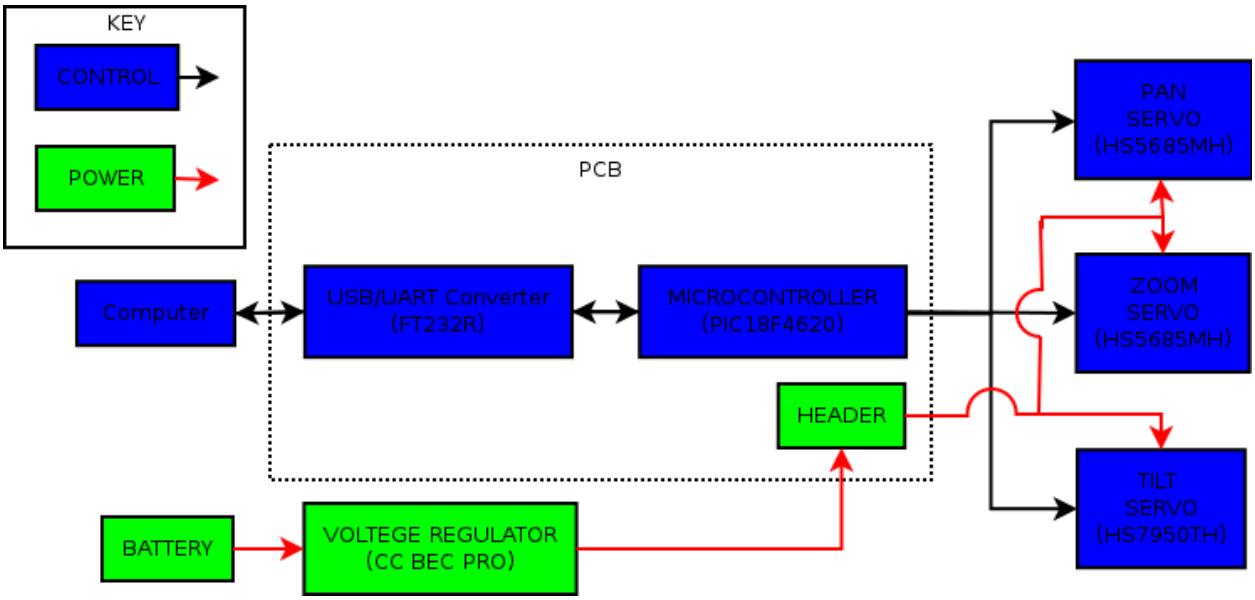


Figure 4.1

PCB

Advanced Circuits manufactured the PCB for our controller. We were able to design our PCB with only two layers and zero vias. The board was designed to fit underneath the pan system with motherboard standoffs. The board and all of its major components are labeled below in figure 5.1. Figure 5.3 shows the thermal performance of the board, taken form a thermal imaging camera. Figure 5.2 shows the dimensions and the traces of the board.

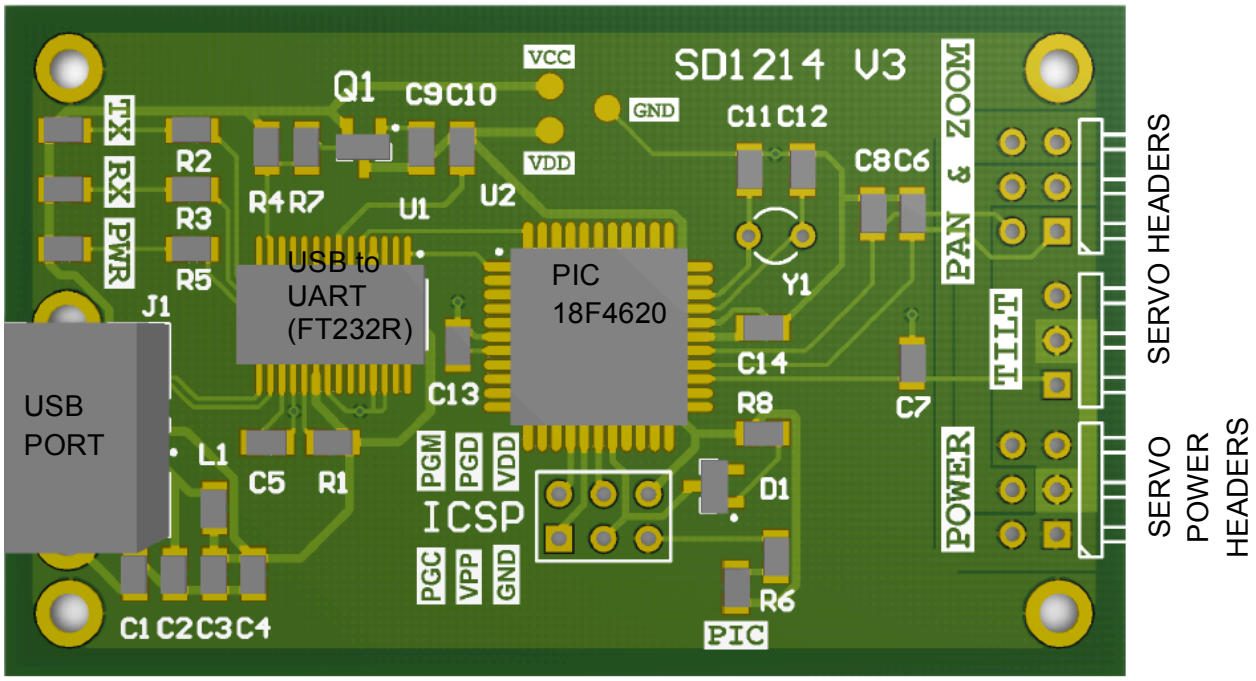


Figure 5.1

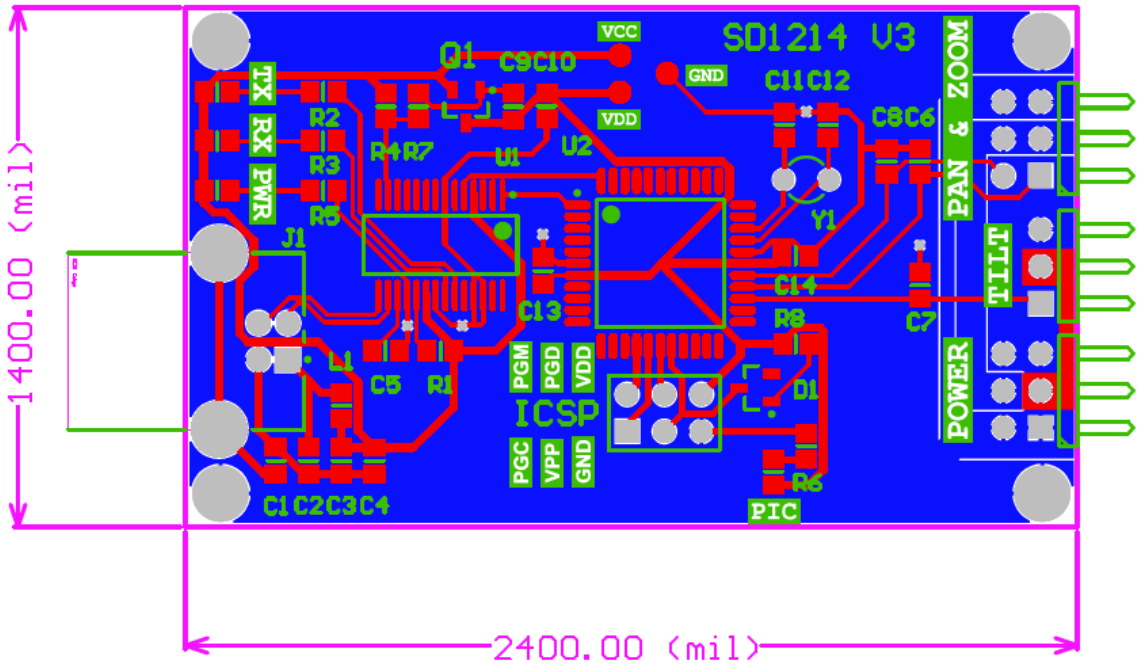


Figure 5.2

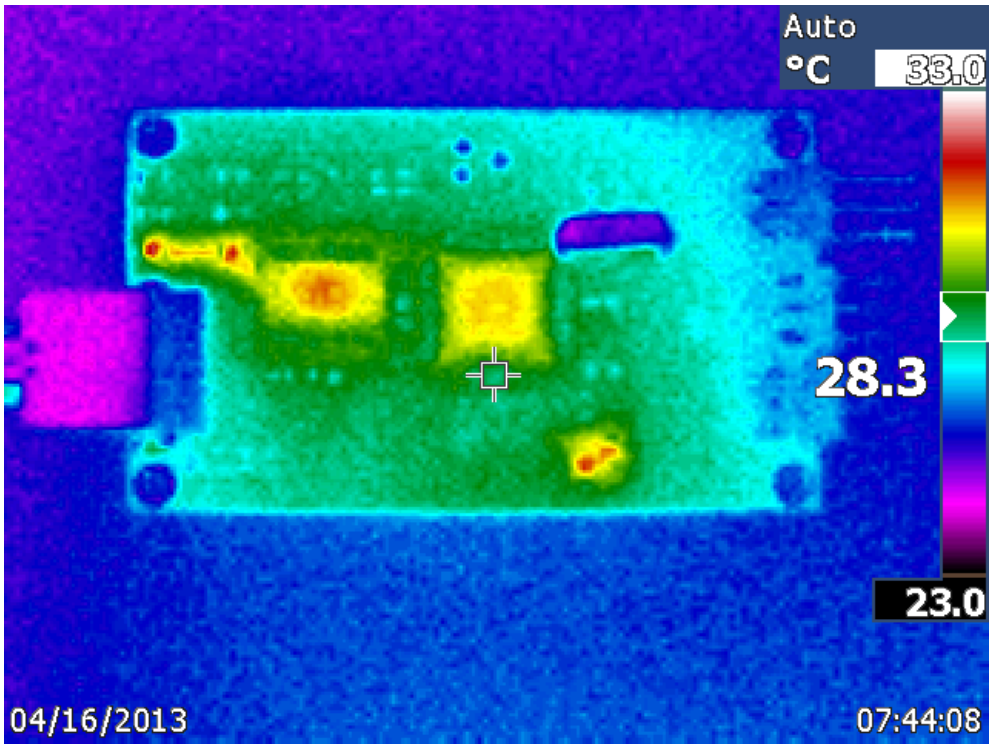
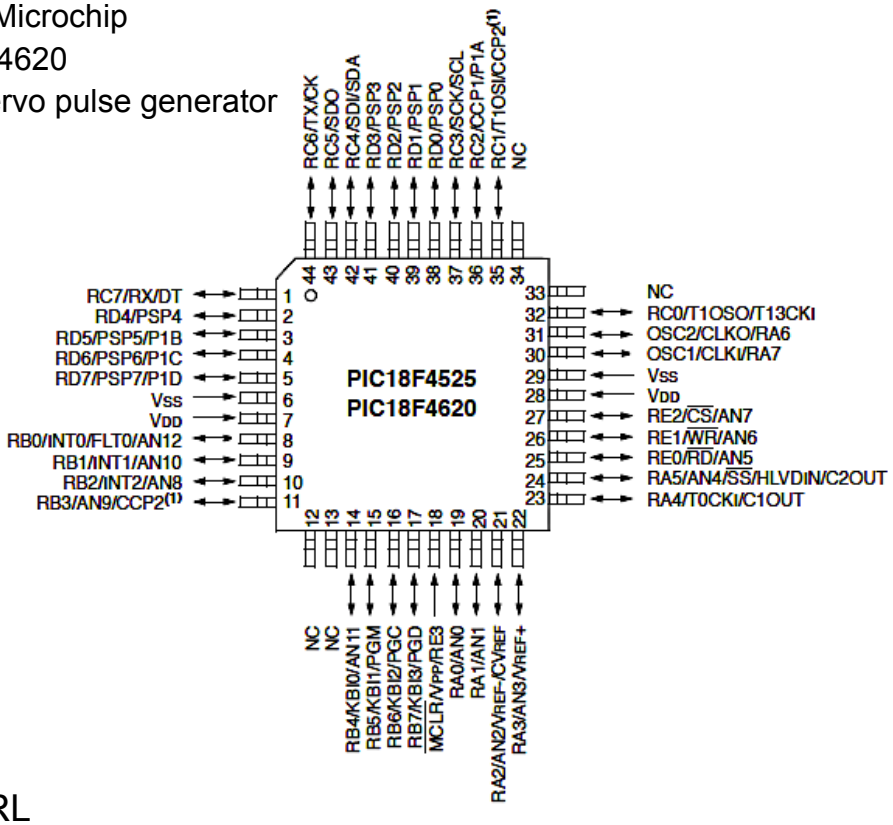


Figure 5.3

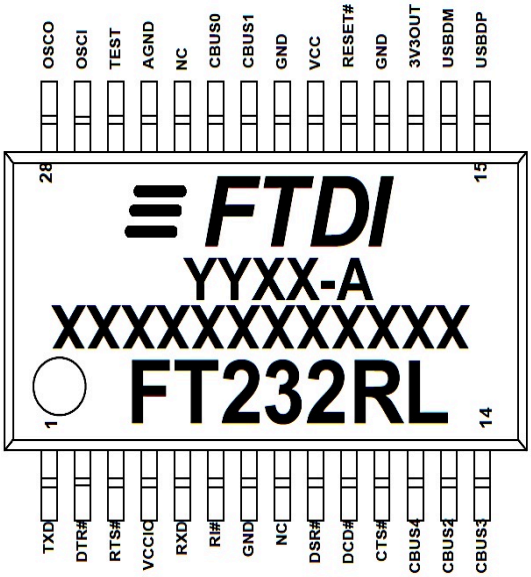
PIC 18F4620

Datasheet: http://ww1.microchip.com/downloads/en/devicedoc/39626b.pdf
Manufacturer: Microchip
Part #: PIC18F4620
Description: Servo pulse generator



FTDI FT232RL

Datasheet: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf
Manufacturer: Future Devices Technology International
Part #: FT232RL
Description: Single chip USB to asynchronous serial data transfer interface



BEC PRO

Datasheet: http://www.castlecreations.com/support/documents/cc_bec_pro_user_guide.pdf
Manufacturer: Castle Creations
Part #: CC BEC Pro
Description: Servo switching voltage regulator

HS 7950TH SERVO

Datasheet: http://www.servocity.com/html/hs-7950th_servo.html
Manufacturer: Hitec Servos
Part #: HS7950TH
Description: Tilt Servo

HS 5685MH SERVOS

Datasheet: http://www.servocity.com/html/hs-5685mh_servo.html
Manufacturer: Hitec Servos
Part #: HS5685MH
Description: Pan and zoom servos

SOFTWARE

The software for this project is divided into two areas, the software that runs on the PIC microcontroller, and the software that runs on the controlling computer.

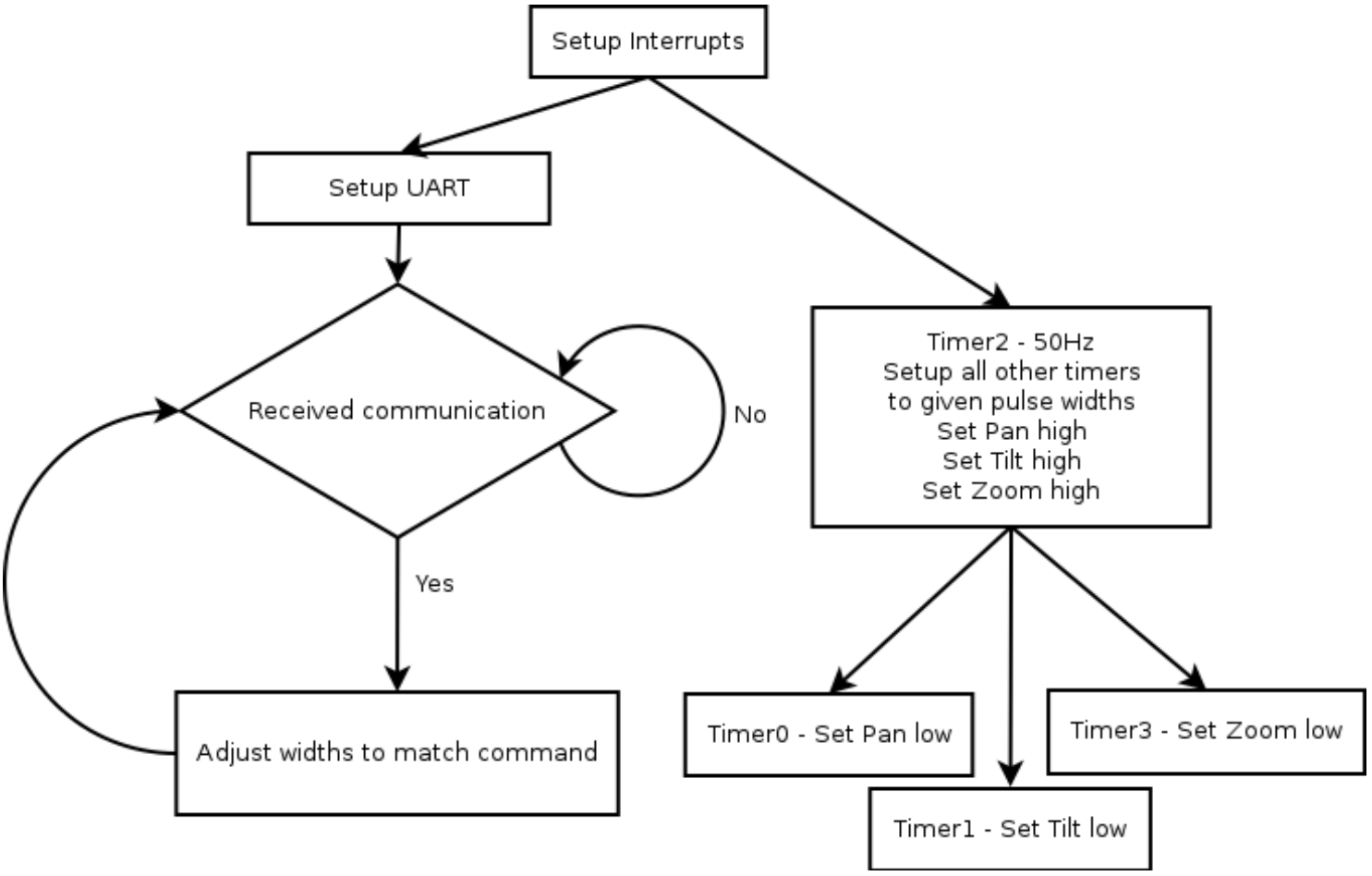
PIC SOFTWARE

The software on the PIC manages generating the 3 output pulse-widths while also communicating with the computer in order to change those pulse-widths to affect where the camera is pointing.

The PIC software runs two separate processing threads at once; the first thread handles communication with the computer. It reads data coming off of the UART line and then adjusts the pulse-width values in memory accordingly.

The other thread uses all 4 of the timers on the PIC, the 8-bit timer setup to get a 50Hz refresh update that then sets the other 3 16-bit timers durations up and raises all the pulse-width pins to a high logic level. Then, when the 16-bit timers fire they bring their respective pin low. This allows for a very tight resolution (5 clock cycles of the PIC or 1us) on 3 pins simultaneously.

PIC SOFTWARE FLOW CHART



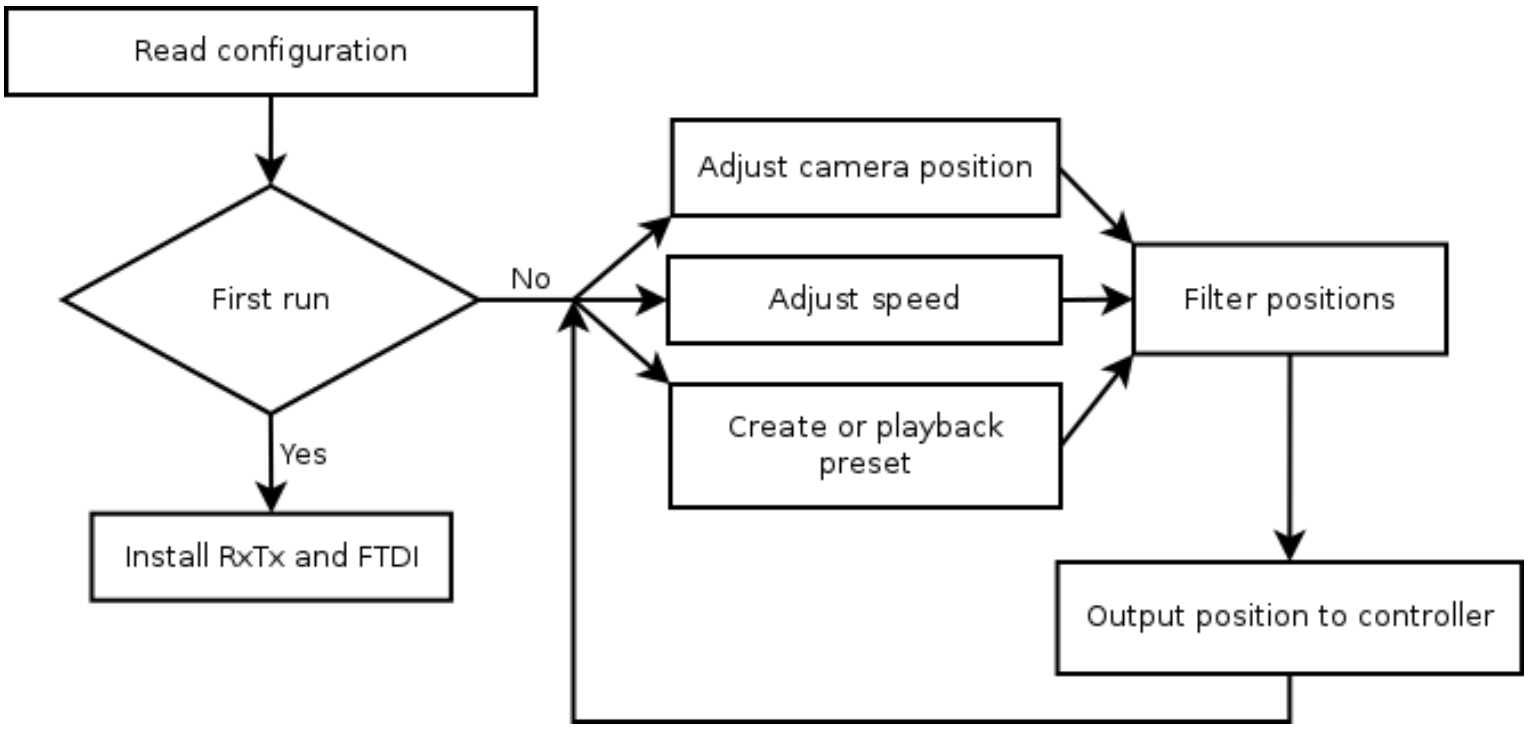
Computer Software

The computer software runs on the client's MacBook Pro, and allows for the commands to be sent to the PIC over USB that change the servomotor angles. The camera in-turn relays the image back to computer to allow the user to see how the changes have affected their shot. The software was designed with our client in mind, so it is as controllable from hotkeys as is possible. The software has a 3-stage process for setup. First it installs as much of itself as it can automatically, and directs the user how to install the FTDI library. Next, it sets up the connection with the PIC, allowing the user to select which serial port is the controller. Finally, it enters into the control screen, which allows full control of the camera. From here all 3-movement planes can be controlled, as well as the playback and setup of preset motions to allow the client to configure some default actions that they would use commonly.

The software sends out commands to the control board on an adjustable update rate, and all smoothing and filtering is done on the user computer. This allows for precise movement times to be setup, and an adjustable rate allows for an optimal rate to be chosen for a given rise time, minimizing jerkiness in the servomotor movement.

An adjustable control filter accomplishes the smoothness. The filter was designed to allow for both change in rise-time and change in sample rate of the control. The idea is that the filter can change the location of its dominant pole, which thereby changes the rise time of its step response. By putting the commanded servo positions through the filter, the speed at which the servomotor adjusts its angle can be controlled precisely, allowing for precision control of the movement of the camera.

Computer Software Flow Chart



TROUBLESHOOTING

Servos won't respond

- Check servos for power
- Check USB connection settings in the application
- Disconnect and reconnect USB cable
- Restart computer

Application won't start

- Disconnect al USB peripherals except the controller
- Restart Computer

PROJECT COMMENTS

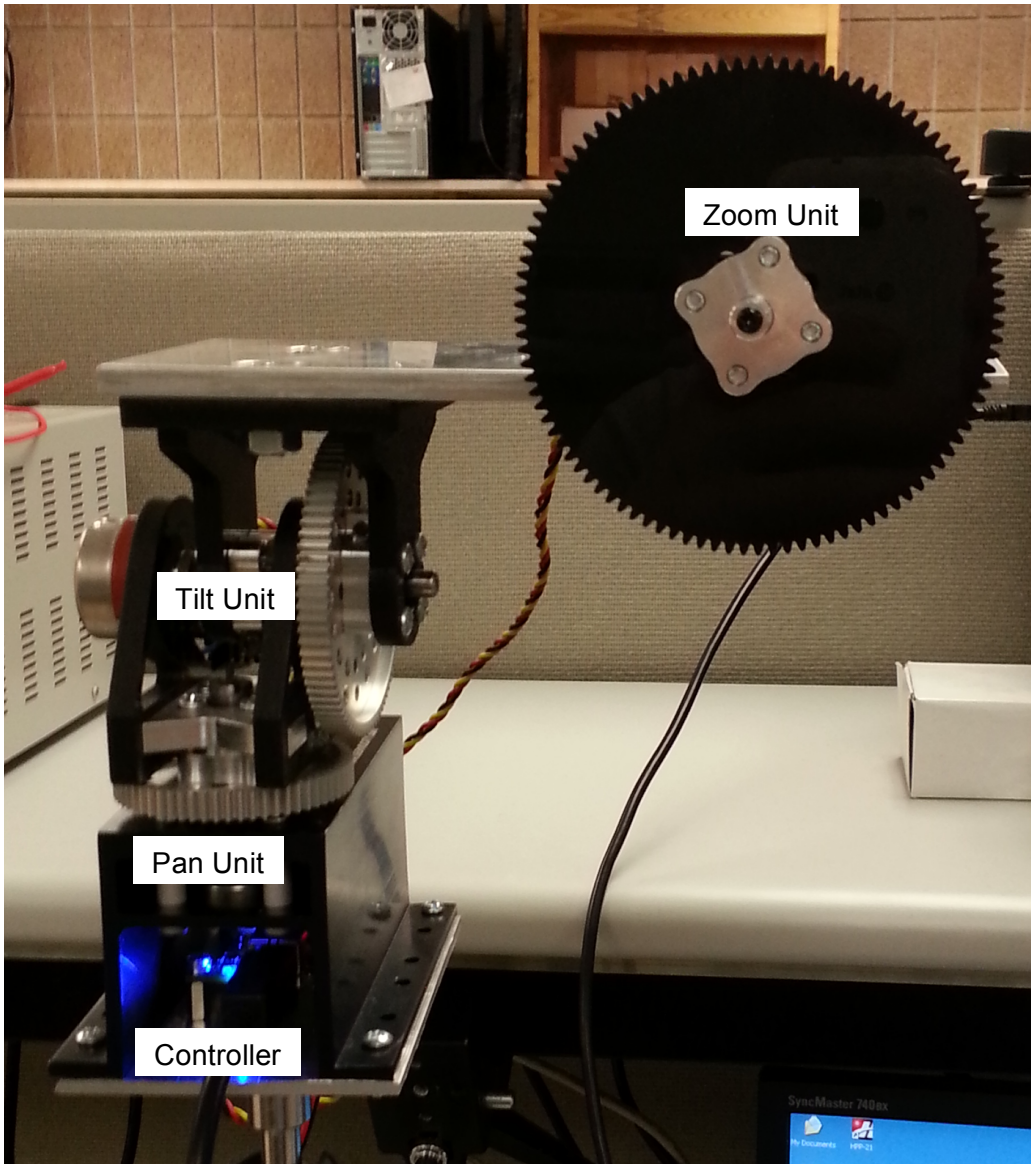
After encountering some speed issues with the microcontroller, we wished that we used a more powerful controller such as a DSPIC. A lesson to be learned form this is to not be afraid of using controllers or components that you don't have experience with. Another lesson we learned is to more clearly define the roles of the members of the group. This helps to make the project more organized, and it also gives the members of the group a more equal share of work. Otherwise, our project went very smooth. We met all of our major deadlines, and more importantly, we were able to meet and exceed our client's expectations.

APPENDIX

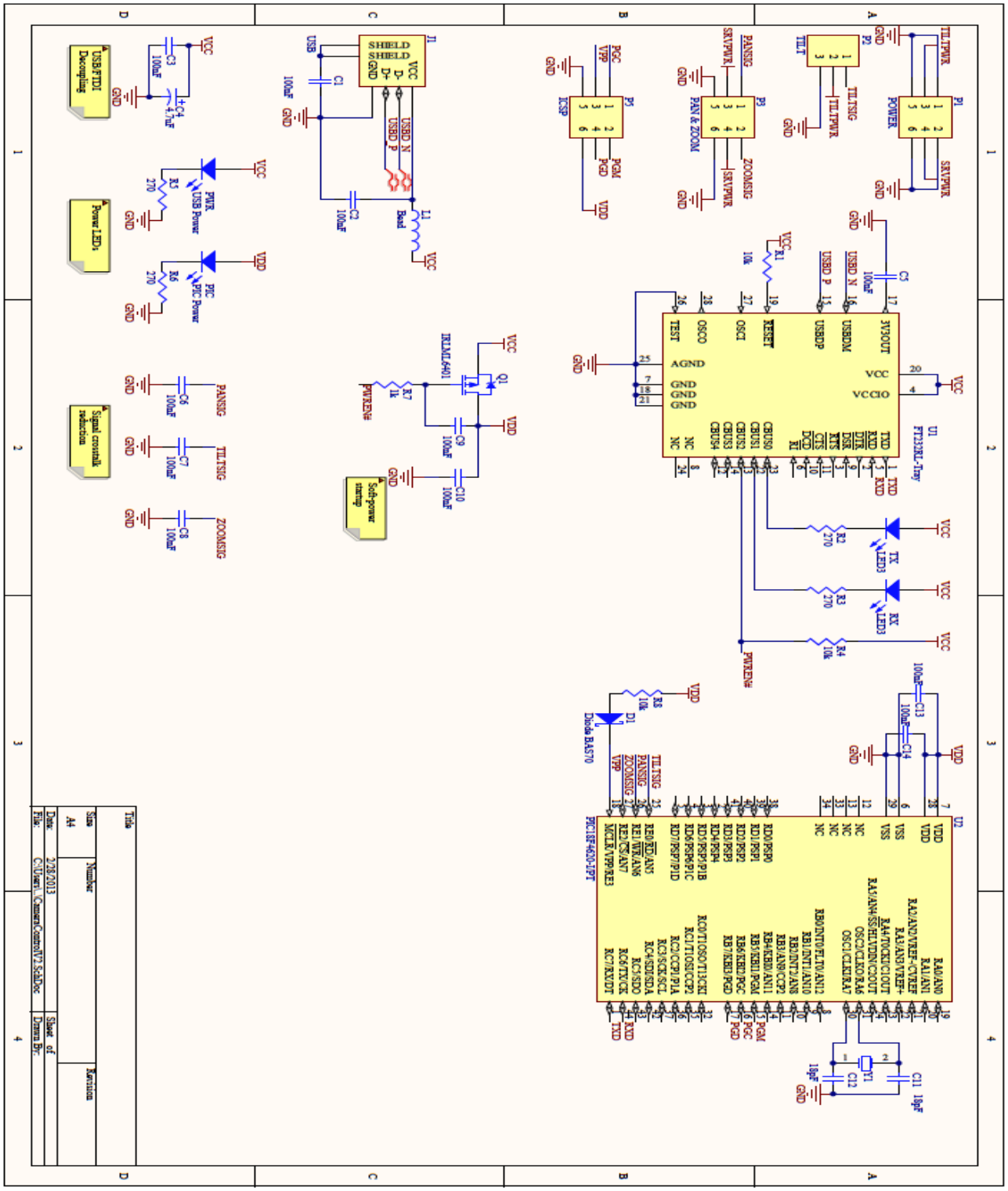
BUDGET

Item ID	Description	Vendor	Cost	Qty.	Total Cost
A31725-ND	USB-B Socket	DigiKey	1.92	2	$3.84
Various	0805 Resistors 270, 1k, 10k	DigiKey	0.01	30	$0.30
Various	0805 Capacitors 100nF	DigiKey	0.01	20	$0.20
IRLML6401PBFCT-ND	IRLML6401 SOT23	DigiKey	0.53	2	$1.06
PIC18F4620-I/PT-ND	PIC18F4620 TQFP-44	DigiKey	7.8	2	$15.60
768-1007-1-ND	FTDI 232RL SSOP	DigiKey	4.5	2	$9.00
	BAS70 Diode	DigiKey	0.45	2	$0.90
PCB Manufacture		Advanced Circuits	50	2	$100.00
SPT400	Tilt System	Servo City	$219.98	1	$219.98
SPG5685A-BM	Pan System	Servo City	$129.98	1	$129.98
HS-5685MH	High torque servo	Servo City	$39.99	3	$119.97
HP-21+	Servo Programmer	Castle Creations	$49.99	1	$49.99
Voltage regulator	010-004-01	Castle Creations	$44.95	1	$44.95
Pulley	10T-6MM-PL	Servo City	$16.99	1	$16.99
15" Belt	B375-150XL	Servo City	$8.99	1	$8.99
Hub Gear	MG64T-32P-250F-50B	Servo City	$12.99	1	$12.99
Drive gear	RSA32-HMG-32	Servo City	$14.99	1	$14.99
USB Programmer	010-0005-00	Servo City	$25.00	1	$25.00
Machining		NDSU IME	$130.00	1	$130.00
Total:					$904.73

DEVICE PICTURE

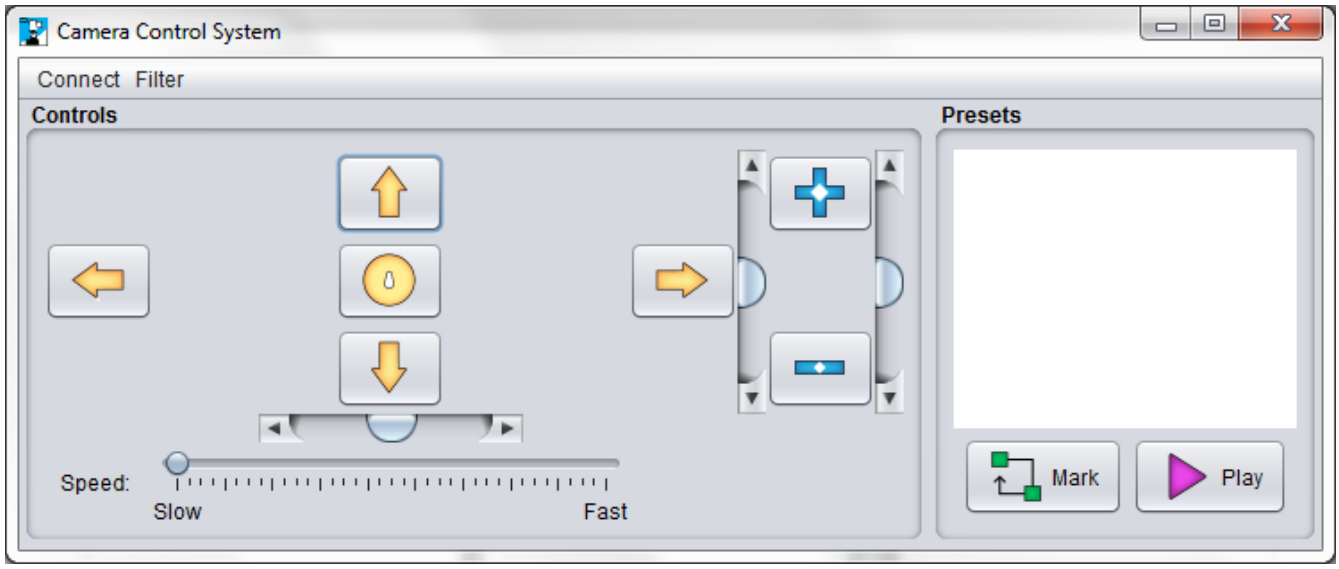


SCHEMATICS



Assistive Camera Control

SOFTWARE GUI



APPLICATION SOFTWARE

*See USB drive

PIC CODE

```
// ******************************************************
// Entry-point file
//
// Contains the entry point for the system, with
// all of the main handling
//
// Author: Steven Goldade
// Last Updated: 01/22/2013
// ******************************************************

#include <htc.h>
#include "protocol.h"
#include "eusart.h"
#include "string.h"

// Pulse width values in microseconds
#define PWM_MIN 900
#define PWM_MID 1500
#define PWM_MAX 2100

// This is the number of clock cycles in 400Hz,
// which is then scaled by 8 to become 50Hz for Timer1
#define HZ_50 12500

// definition of which pins correspond to the servo signals
#define PAN_PIN LATE1
#define TILT_PIN LATE0
#define ZOOM_PIN LATE2

// variables used for PWM
unsigned int pan_width = 0;
unsigned int tilt_width = 0;
unsigned int zoom_width = 0;

unsigned char tmr2_count = 0;

void center();
```

```
/**
 * Interrupt Service Routine, handles all interrupts
 */
void interrupt ISR() {
	//EUSART
	EUSART_ISR();

	// Timer2 - 50Hz
	if(TMR2IF) {
		if(tmr2_count < 4) {
			tmr2_count++;
		} else if(tmr2_count == 4) {
			// Timer2 setup for 160 clocks ideally, but due to time in calculation it is different
			T2CON = 0x07;
			PR2 = 97;
			tmr2_count++;
		} else {
			// 50Hz tick
			TMR0 = -5*pan_width;
			TMR1 = -5*tilt_width;
			TMR3 = -5*zoom_width;

			PAN_PIN = 1;
			TMR0ON = 1;

			TILT_PIN = 1;
			TMR1ON = 1;

			ZOOM_PIN = 1;
			TMR3ON = 1;

	//Timer 2 - ~250Hz ideally, but setup here is different due to time that timer2 spends in calculation
			T2CON = 0x77;
			PR2 = 81;
			tmr2_count = 0;
		}
		TMR2IF = 0;
	}

	// Timer0 - Pan
	if(TMR0IF) {
```

```
                PAN_PIN = 0;

                TMR0ON = 0;
                TMR0IF = 0;
        }

        // Timer1 - Tilt
        if(TMR1IF) {
                TILT_PIN = 0;

                TMR1ON = 0;
                TMR1IF = 0;
        }

        // Timer3 - Zoom
        if(TMR3IF) {
                ZOOM_PIN = 0;

                TMR3ON = 0;
                TMR3IF = 0;
        }
}

/**
 * Sets up and initializes interrupts
 */
void initInterrupt() {
        // Timer2 setup, ~250Hz
        T2CON = 0x77;
        PR2 = 81;
        TMR2IE = 1;
        TMR2IP = 1;

        // Timer0 setup, no prescaler
        T0CON = 0x08;
        TMR0IE = 1;
        TMR0IP = 1;

        // Timer1 setup, no prescaler
        T1CON = 0x80;
        TMR1IE = 1;
```

```
	TMR1IP = 1;

	// Timer3 setup, no prescaler
	T3CON = 0x80;
	TMR3IE = 1;
	TMR3IP = 1;

	// default all servos to center position
	center();

	// enable interrupts
	PEIE = 1;
	GIE = 1;
}

/**
 * Initializes the microcontroller
 */
void initialize() {
	// default all ports to output and output low
	TRISA = 0;
	TRISB = 0;
	TRISC = 0;
	TRISD = 0;
	TRISE = 0;
	LATA = 0;
	LATB = 0;
	LATC = 0;
	LATD = 0;
	LATE = 0;

	// initialize EUSART module and interrupts
	EUSART_Init();
	initInterrupt();
}

/**
 * Attemps to acquire the baud rate, or defaults to 9600 if it fails
 */
void getBaudRate() {
	char succeed = EUSART_GetAutoBaudRate();
```

```
		if(succeed) {
			EUSART_Transmit(SYNC_RECV);
		} else {
			// default to 9600
			EUSART_SetBaudRate(9600L);
			EUSART_Transmit(SYNC_RECV);
		}
}

/**
 * Change the pan pulsewidth to val microseconds
 */
void changePan(unsigned int val) {
		if(val < 900) {
			val = 900;
		} else if(val > 2100) {
			val = 2100;
		}
		pan_width = val;
}

/**
 * Change the tilt pulsewidth to val microseconds
 */
void changeTilt(unsigned int val) {
		if(val < 900) {
			val = 900;
		} else if(val > 2100) {
			val = 2100;
		}
		tilt_width = val;
}

/**
 * Change the zoom pulsewidth to val microseconds
 */
void changeZoom(unsigned int val) {
		if(val < 900) {
			val = 900;
		} else if(val > 2100) {
			val = 2100;
```

```
		}
		zoom_width = val;
}

/**
 * Centers the pulsewidth to all servos
 */
void center() {
		pan_width = PWM_MID;
		tilt_width = PWM_MID;
		zoom_width = PWM_MID;
}

/**
 * Handles all protocol and communication
 */
void handleProtocol() {
		unsigned char msg[32];
		unsigned char* msgData = &msg[1];

		EUSART_Receive(msg);
		if(strlen(msg) > 0) {
			EUSART_TransmitMsg(msg);
			switch(msg[0]) {
				case CHANGE_PAN:
					changePan(toInt(msgData));
					break;
				case CHANGE_TILT:
					changeTilt(toInt(msgData));
					break;
				case CHANGE_ZOOM:
					changeZoom(toInt(msgData));
					break;
				case CENTER:
					center();
					break;
				case SYNC_SEND:
					EUSART_Transmit(SYNC_RECV);
					break;
				case POSITION: {
					// build position information
```

```
                                        unsigned char pan[6];
                                        unsigned char tilt[6];
                                        unsigned char zoom[6];
                                        toString(pan_width, pan);
                                        toString(tilt_width, tilt);
                                        toString(zoom_width, zoom);

                                        pan[0] = CHANGE_PAN;
                                        tilt[0] = CHANGE_TILT;
                                        zoom[0] = CHANGE_ZOOM;
                                        // send position information
                                        EUSART_TransmitMsg(pan);
                                        EUSART_TransmitMsg(tilt);
                                        EUSART_TransmitMsg(zoom);
                                        break;
                                }
                        }
                }
}

/**
 * Main routine, Entry point and operational loop
 */
void main() {
        // setup the microcontroller
        initialize();
        getBaudRate();

        // begin operation
        while(1) {
                // handle communications and respond
                handleProtocol();
        }
}

// ******************************************************
// EUSART implementation file
//
// Implements the EUSART module
//
// Author: Steven Goldade
```

```
// Last Updated: 01/22/2013
// ******************************************************

#include <htc.h>
#include "protocol.h"
#include "eusart.h"
#include "string.h"

// defines the communication delimiter
#define DELIMETER CR
// defines the buffer size
#define BUFFER_SIZE 32

// the previously buffered received message
unsigned char RXMSG[BUFFER_SIZE];

// the currently buffered message
unsigned char MSG[BUFFER_SIZE];
unsigned char MSG_LENGTH = 0;

// a flag indicating whether a complete message has
// been received
unsigned char RX_FLAG = 0;

// a flag indicating if the auto baud rate detection
// is currently active
unsigned char ABD_FLAG = 0;

void EUSART_ISR() {
	if(RCIF) {
		// read current byte
		unsigned char rxChar = RCREG;

		if(rxChar == DELIMETER) {
			// Received a delimeter, this message is completed
			MSG[MSG_LENGTH] = NULL_CHAR;
			strncpy(MSG, RXMSG, MSG_LENGTH);
			MSG_LENGTH = 0;
			RX_FLAG = 1;
		} else {
			// Normal character
```

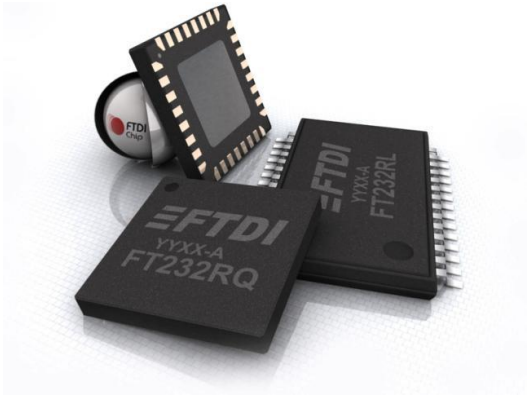
```
                        MSG[MSG_LENGTH] = rxChar;
                        MSG_LENGTH++;
                        if(MSG_LENGTH == BUFFER_SIZE) MSG_LENGTH = BUFFER_SIZE-1;
                }

                // if we are currently auto-detecting the baud rate
    // trash any received data
                if(ABD_FLAG) {
                        MSG_LENGTH = 0;
                }

                // if an error occurs during receiving, attempt
    // to broadcast that there was an error
                if(OERR) {
                        EUSART_Transmit(RECV_ERR);
                        CREN = 0;
                        CREN = 1;
                }
        }
}

void EUSART_Init() {
        TXSTA = 0x26; // Async, high baud
        RCSTA = 0x90; // Turn on receiver, Async
        BAUDCON = 0x80; // 16-bit baud generation

        TRISCbits.RC7 = 1; // enable RX
        RCIE = 1; // enable RX interrupt
}

void EUSART_Transmit(unsigned char data) {
        while(!TRMT); // wait for any current transmission to finish
        TXREG = data;
}

void EUSART_TransmitMsg(unsigned char* src) {
        for(unsigned char p=0; p<strlen(src); p++) {
                while(!TRMT); // wait for any current transmission to finish
                TXREG = src[p];
        }
        while(!TRMT);
```

```
	TXREG = DELIMETER;
	while(!TRMT);
}

void EUSART_Receive(unsigned char* dest) {
	if(RX_FLAG) {
		RX_FLAG = 0;
		strcpy(RXMSG,dest);
	} else {
		*dest = NULL_CHAR;
	}
}

char EUSART_GetAutoBaudRate() {
	ABD_FLAG = 1;

	ABDEN = 1; // enable auto baud rate detection
	while(ABDEN); // wait for detection to finish

	ABD_FLAG = 0;

	// error returns -1, 1 else-wise
	if(ABDOVF) {
		return -1;
	} else {
		return 1;
	}
}

void EUSART_SetBaudRate(unsigned long baudRate) {
	BRGH = 1; // sets 16 bit baud rate
	BRG16 = 1;
	SYNC = 0;

	// calculates SPBRG
	unsigned long regval = 5000000/baudRate - 1;
	SPBRG = (unsigned int)regval;
}
```

DATASHEET COVER PAGES

Future Technology Devices International Ltd.

FT232R USB UART IC



The FT232R is a USB to serial UART interface with the following advanced features:

- Single chip USB to asynchronous serial data transfer interface.
- Entire USB protocol handled on the chip. No USB specific firmware programming required.
- Fully integrated 1024 bit EEPROM storing device descriptors and CBUS I/O configuration.
- Fully integrated USB termination resistors.
- Fully integrated clock generation with no external crystal required plus optional clock output selection enabling a glue-less interface to external MCU or FPGA.
- Data transfer rates from 300 baud to 3 Mbaud (RS422, RS485, RS232) at TTL levels.
- 128 byte receive buffer and 256 byte transmit buffer utilising buffer smoothing technology to allow for high data throughput.
- FTDI's royalty-free Virtual Com Port (VCP) and Direct (D2XX) drivers eliminate the requirement for USB driver development in most cases.
- Unique USB FTDIChip-ID™ feature.
- Configurable CBUS I/O pins.
- Transmit and receive LED drive signals.
- UART interface support for 7 or 8 data bits, 1 or 2 stop bits and odd / even / mark / space / no parity
- FIFO receive and transmit buffers for high data throughput.
- Synchronous and asynchronous bit bang interface options with RD# and WR# strobes.
- Device supplied pre-programmed with unique USB serial number.
- Supports bus powered, self powered and high-power bus powered USB configurations.
- Integrated +3.3V level converter for USB I/O.
- Integrated level converter on UART and CBUS for interfacing to between +1.8V and +5V logic.
- True 5V/3.3V/2.8V/1.8V CMOS drive output and TTL input.
- Configurable I/O pin output drive strength.
- Integrated power-on-reset circuit.
- Fully integrated AVCC supply filtering - no external filtering required.
- UART signal inversion option.
- +3.3V (using external oscillator) to +5.25V (internal oscillator) Single Supply Operation.
- Low operating and USB suspend current.
- Low USB bandwidth consumption.
- UHCI/OHCI/EHCI host controller compatible.
- USB 2.0 Full Speed compatible.
- -40°C to 85°C extended operating temperature range.
- Available in compact Pb-free 28 Pin SSOP and QFN-32 packages (both RoHS compliant).

PIC18F2525/2620/4525/4620
Data Sheet

28/40/44-Pin
Enhanced Flash Microcontrollers
with 10-Bit A/D and nanoWatt Technology

PIC18F2525/2620/4525/4620
Data Sheet

28/40/44-Pin
Enhanced Flash Microcontrollers
with 10-Bit A/D and nanoWatt Technology

PIC18F2525/2620/4525/4620
Data Sheet

28/40/44-Pin
Enhanced Flash Microcontrollers
with 10-Bit A/D and nanoWatt Technology